DeepEye: Visualizing Your Data by Keyword Search

[Visionary Paper]

Xuedi Qin
Department of Computer Science, Tsinghua University
qxd17@mails.tsinghua.edu.cn

Yuyu Luo
Department of Computer Science, Tsinghua University
luoyuyu@mail.tsinghua.edu.cn

Nan Tang
Qatar Computing Research Institute, HBKU
ntang@hbku.edu.qa

Guoliang Li*
Department of Computer Science, Tsinghua University
liguoliang@tsinghua.edu.cn

ABSTRACT

Do you dream to create good visualizations for your dataset simply like a Google search? If yes, our visionary system DeepEye is committed to fulfill this task. Given a dataset and a keyword query, DeepEye understands the query intent, generates and ranks good visualizations. The user can pick the one he likes and do a further faceted search to easily navigate the visualizations. We detail the architecture of DeepEye, key components, as well as research challenges and opportunities.

1 INTRODUCTION

Nowadays, the ability to create good visualizations has shifted from a nice-to-have skill to a must-have skill for all data analysts. However, the overwhelming choices of interactive data visualization tools (e.g., Tableau, Microsoft Excel and D3 [5]) only allow experts to create good visualizations, assuming that the experts know many details: the meaning and the distribution of the data, the right combination of attributes, and the right type of charts – these requirements are apparently not easy, even for experts.

Unfortunately, creating good data visualization is hard. From the user perspective, there are many possible ways of visualizations for a given dataset (for example, different attribute combinations and visualization types), and many ways of transforming data (for example, grouping, binning, sorting, and a combination thereof) – these make it infeasible for the user to enumerate all possible visualizations and select the ones he needs. From the system perspective, among numerous problems, no consensus has emerged to quantify the "goodness" of a visualization. What makes it harder is when the system does not even know what the user wants. Recently, there have been proposals for visualization recommendation systems [15, 18], which focus on automatically recommending "interesting" visualizations from a diversified criteria, such as relevance, surprise, non-obviousness, diversity and coverage. However, as pointed out by [3], these systems may mislead the user, by generating visualizations that might be *worse than nothing*, since it is basically impossible to guess a user's query intent from nothing.

The natural problem arises: *What is the most feasible way to (automatically) create good visualizations even for dummies?*

We have three key intuitions for handling the above problem.

(1) *What is the ideal language for the user to specify his intent?* Of course, the mother tongue – Google-like natural language search interface is friendly to everyone. Note that (i) most visualizations are generated using declarative languages like Vega-Lite [12]; and (ii) using machine learning to convert text to SQL queries for relational databases has been proved effective [11]. Hence, we can allow users to pose keyword queries and our system automatically converts them to declarative visualization queries.

Keyword-to-Visualizations is hard: keywords are typically ambiguous and underspecified; and good visualizations concern statistical properties such as trends, comparisons, which are rather hard to grasp even for human.

(2) *How can we solve the fundamental cognitive science problem for quantifying* good *visualizations?* Evidently, good visualizations exist, which can be collected from many sources that take experts hours or even days to produce such valuable visualizations. Naturally, the research problem is how to transfer the knowledge from these known good visualizations to judge a new visualization.

Transferring good visualizations. The basic intuition is: a new visualization is good, if it *matches* some known good visualization (see Section 2.2 for a further discussion).

(3) *How can we rank visualizations?* Ranking is the secret sauce to any search engine. Note, however, that it is almost impossible to rank visualizations, if they are independent of each other.

Visualization link graph. We propose a novel graphical model to link good visualizations. Informally speaking, two visualizations are linked if they are *relevant*, which are further classified into different types (or facets), *e.g.,* similar or diverse that are captured by different facet functions (or criteria [15]).

The main benefit of having the above graph is twofold. (i) We can devise *PageRank* [6] like algorithms to rank visualizations. (ii) We can leverage the edge types to provide faceted search (*a.k.a.* faceted browsing) such that the user can easily set the compass to navigate the ocean of visualizations by simple clicks.

DeepEye is our visionary system to create good visualizations by *keyword search and simple clicks*.

DeepEye **Workflow.** A user can pose a keyword query K. DeepEye translates this keyword query to multiple candidate visualizations $\mathbf{V}$, discovers good visualizations $\mathbf{V'}$ in $\mathbf{V}$, ranks $\mathbf{V'}$, and returns the top ones $\mathbf{V''}$. When the user selects a visualization V in $\mathbf{V''}$ and further explores by clicking a facet of V, DeepEye discovers more visualizations and helps the user to easily explore his desired visualizations. The user may iterate over the above processes until he finds all visualizations that he wants.

Remarks. (1) Different from *visualization recommendation systems* [15, 18], (i) DeepEye is a visualization search engine that the user needs to provide his intent – we do not believe in the magic that one can guess something from nothing; and (ii) DeepEye decides and ranks good visualizations by comparing with known good visualizations. (2) After finding good visualizations,

Guoliang Li is the Corresponding Author.
© 2018 Copyright held by the owner/author(s). Published in Proceedings of the 21st International Conference on Extending Database Technology (EDBT), March 26-29, 2018, ISBN 978-3-89318-078-3 on OpenProceedings.org.
Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

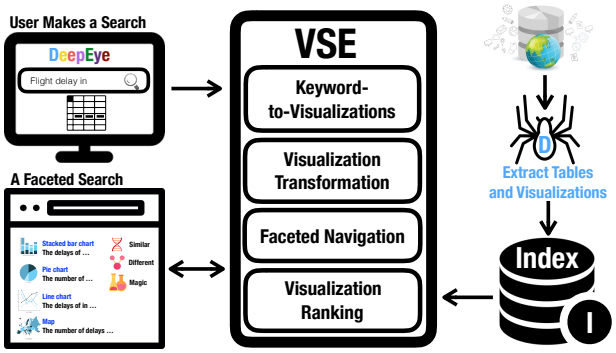

Figure 1: The Architecture of DeepEye

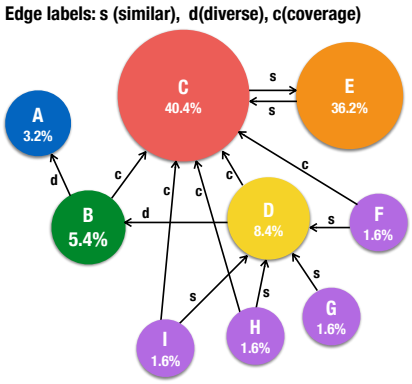

Figure 2: A Sample Visualization Link Graph

DeepEye will support to export them to a designated interactive data visualization tool (*e.g.,* Tableau) for more customized manipulations. (3) Although intuitively, DeepEye, as a search engine, shares a similar architecture with well known search engines such as Google and Bing, there are many new challenges for designing visualization search algorithms (see Section 3).

2 THE DEEPEYE ARCHITECTURE

The architecture of DeepEye is given in Figure 1. DeepEye crawls, stores and indexes good visualizations from multiple sources.

The user starts by posing a *keyword search* and provides a dataset. The *visualization search engine* (VSE) will first generate a set of visualizations $\mathbf{V}$ by the *Keyword-to-Visualizations* module. These candidate visualizations $\mathbf{V}$ will be matched with known good visualizations by the *Visualization Transformation* module that produces $\mathbf{V}' \subseteq \mathbf{V}$, which will then be ranked by the *Visualization Ranking* module and the top ones $\mathbf{V}'' \subseteq \mathbf{V}'$ are returned to the user. The user may pick the one he likes and discovers more visualizations by the *Faceted Navigation* module. This module aims to reduce the number of user interactions and helps users to find the target visualizations as soon as possible.

2.1 Preliminary

Datasets. We crawl visualizations with data and visualizations charts (e.g., pie/line/bar charts) from multiple sources. Then we use them to visualize a user-given dataset D. For simplicity, we consider D as only one table, which can be easily extended to support multiple tables by relational joins (see SeeDB [16]).

Data Features. Typically, what will decide whether a visualization of a dataset is good or not depends more on its features (or representations [2]), not its data values. More specifically, we consider the following features of a dataset D: the data type of a column (*e.g.,* categorical, numerical, and temporal), the number of distinct values of a column, the number of tuples in a column, the ratio of unique values in a column, the max() and min() values of a column, and statistical correlation between two columns (*e.g.,* linear, polynomial, power, and log).

Visualization Queries. For *declarative visualization language*, we use Vega-Lite [12], a high-level grammar that enables rapid specification of interactive data visualizations. (Note that our system can support any declarative visualization query language.) Each *query* Q, specified in the Vega-Lite JSON format over the dataset D, denoted by $Q(D)$, will produce a visualization. A sample Vega-Lite query is as follows:

```
"data": {"url": "flights.json"},
"mark": "bar",
"encoding": {
    "x": {
        "bin": true,
        "field": "carriers",
        "type": "qualitative" },
    "y": {
        "aggregate": "count",
        "type": "quantitative" },
}
```

Visualization Crawler. The DeepEye *crawler* will extract tables and their associated visualizations from multiple sources, where *both* the table *and* the visualization specifications (or equivalently, queries) need to be explicitly given. For example, there are hundreds of visualization examples in https://www.highcharts.com, with both data values and visualization specifications. Note that we do not require these sources to use Vega-lite – most visualization specifications can be easily converted to Vega-lite queries, for which we need to implement corresponding tools for query rewriting. When only the data and visualizations are given but the visualization queries are absent, an interesting open problem is to automatically infer the declarative visualization queries.

Visualization Link Graph. A *visualization link graph* is a directed graph $\mathbf{G}(\mathbf{N}, \mathbf{E})$ with nodes $\mathbf{N}$ and directed edges $\mathbf{E}$. Each node $v \in \mathbf{N}$ is a visualization (*i.e.,* $Q(D)$). Each directed edge $e : (u, v) \in \mathbf{E}$ can have multiple labels, denoted by $\mathbf{L}(e)$, where each label $L(e) \in \mathbf{L}(e)$ is a facet, such as similar, diverse, coverage.

In order to decide the edges and their labels, we have defined a set of *facet functions*. For example, there is a "similar"-edge from node u to node v, if the corresponding *similar-function* $f_s(u, v)$ returns *true*, denoting that u is similar to v. There can have another "similar"-edge from v to u if $f_s(v, u)$ also returns *true*. Note that any *facet function* does not have to be symmetric – $f(u, v)$ is *true* does not imply that $f(v, u)$ is *true* as well. The other edges and labels are generated similarly using different *facet functions*. Please find more details in the *Faceted Navigation* module in Section 2.2.

A sample visualization link graph is given in Figure 2. The number associated with each node denotes the importance of the node, which will be further discussed in Section 3.

2.2 Visualization Search Engine Modules

In this section, we will discuss the functionalities and our basic designs for the modules of DeepEye visualization search engine.

Keyword-to-Visualizations. Given a keyword query K and a dataset D, this module generates all candidate visualizations.

As mentioned earlier, this problem is hard because keyword queries are always ambiguous and underspecified – there might have a large number of candidate visualizations due to different attribute combinations and multiple data transformation operations (*e.g.,* grouping, binning, sorting). Fortunately, there are simple observations about good/bad visualizations, *e.g.,* pie charts are best to use when comparing parts of a whole, and they do not show changes over time; and bar graphs are used to compare things between different groups or to track changes over time, and too many bars (*e.g.,* > 50) are hard for human to interpret. The traditional wisdom from visualization experts can be encoded into rules to prune many apparently bad charts (see *e.g.,* https://www.pinterest.com/pin/20125529565819990/ for a chart type cheat sheet that can be easily leveraged).

Visualization Transformation. The broad intuition is that, if a visualization V can "match" a known (crawled) good visualization, then V is probably good. As mentioned earlier in the data features, visualization matching focuses more on feature matching (*e.g.,* similar domains and similar trends), in contrast to traditional value matching of strings.

Given a visualization V and another visualization N, it is to compute the similarity between their features, which typically falls in the range $[0, 1]$; and we say that V *matches* N if their similarity is above a threshold σ.

Given a set of visualizations $\mathbf{V}$, the module of *Visualization Transformation* will find a set of existing good visualizations $\mathbf{N}$ that *match* these candidate visualizations. Note that one candidate query can match multiple good visualizations, and vice versa. Also, a candidate visualization that cannot match any existing good one will be removed, which will result in a subset $\mathbf{V}'$ of $\mathbf{V}$.

Visualization Ranking. Given a set of visualizations $\mathbf{V}'$ and their matched good visualizations $\mathbf{N}$, the *Visualization Ranking* module will rank $\mathbf{V}'$ based on $\mathbf{N}$ (*i.e.,* a subgraph of the visualization linkage graph) and return to the user. We can either use the learning-to-rank [7] techniques to learn the features from known good visualizations or design new ranking functions. We can also use the user click-through data to rank the visualizations.

Faceted Navigation. When a user picks a visualization V he likes, he can further explore other visualizations by facets. The *Faceted Navigation* module will discover another set of visualizations based on V, which will also be ranked and returned to the user, in order to help users navigate the visualizations.

We plan to implement this module by providing each facet a programming interface (*i.e.,* an API) that implements a *facet function*, *e.g.,* similar, diverse, coverage. Also, we make it extensible by allowing domain experts to plug in other APIs for different facets. The main reason to allow this flexibility is that till now, there is no consensus about criteria of finding interesting visualizations, which remains an open problem. Note that these facet-functions do not need to be mutually exclusive.

3 RESEARCH OPPORTUNITIES

3.1 Visualization Data

An effective visualization system replies on high-quality and high-coverage visualization data. Although there are some open websites that we can crawl some data, the coverage is limited and we need to construct a visualization benchmark.

Opportunity. First, our system requires visualizations with data, visualization queries and visualization charts. Many websites, however, do not contain such data, and we need to infer one dimension based on the others, *e.g.,* inferring a query based on the data and a visualization. Second, we also need some well-ranked visualizations to help us rank the visualizations for a new dataset. However many websites do not contain rankings, and it is challenging to infer rankings. Third, we can use crowdsourcing to collect more visualization data and rank the visualizations, and the challenge is to reduce the crowdsourcing cost, improve the quality, and avoid the redundancy with existing data.

3.2 Visualization Search Engine

3.2.1 Keyword-to-Visuliazations. The essential problem is to understand natural language and generate visualization queries. Fortunately, the recent machine learning and deep learning techniques have made it easy to understand natural language (see *e.g.,* the OpenNLP toolkit: https://opennlp.apache.org). Furthermore, several approaches have studied the problem of translating natural language to SQL queries, such as NLIDBs [1] and NaLIR [11].

Opportunity. Although the above approaches shed some light on our problem, translating natural languages to visualization queries remains a hard problem. The main reason is that when searching visualizations, the user cares more about the statistical properties such as *trends, comparisons, fast increase*, which are more ambiguous than querying a DBMS such as "return the average number of publications by Bob in each year". In other words, even if an expert knows precisely what is the meaning of the natural language query of the user, it is still hard for the expert to formulate using the declarative visualization language, since good visualizations are also data dependent.

The research problem is: Given a keyword query K and a dataset D, discovers candidate visualizations $\mathbf{Q}(D')$ that the user wants, where D' could be transformed from D (*e.g.,* by operations like binning, grouping, sorting, and a combination thereof).

3.2.2 Visualization Transformation. The fundamental problem of visualization transformation is to transform the knowledge (or features) from known good visualizations to deciding the goodness of unknown visualizations. A possible way is to compute the similarity between two visualizations $Q_1(D_1)$ and $Q_2(D_2)$ (*i.e.,* the known good one). If $Q_1(D_1)$ is very similar to $Q_2(D_2)$ and $Q_2(D_2)$ is good, then by inference, $Q_1(D_1)$ is also good: showing an interesting trend, using the same chart as employed in Q_2, etc.

Opportunity. The open problem is how to define the similarity function $\mathsf{sim}(Q_1(D_1), Q_2(D_2))$, which relates to many factors.

(1) The statistical correlation between the two visualizations $Q_1(D_1)$ and $Q_2(D_2)$, such as the same trend.
(2) The domain similarity, *i.e.,* whether the attributes used for $Q_1(D_1)$ and $Q_2(D_2)$ come from the same domain.
(3) The type similarity, *e.g.,* whether the data types used in $Q_1(D_1)$ and $Q_2(D_2)$ are both temporal data.

3.2.3 Visualization Ranking. As mentioned earlier, there is still no consensus to quantify the "goodness" of a visualization. Intuitively, it is harder to quantify "better" visualizations.

Opportunity. The great success of search engines has shown us multiple ways of doing link analysis, with the purpose of "measuring" the relative importance within a set of linked webpages. Maybe the most successful story is *PageRank* [6]. We have defined

our visualization link graph (see Figure 2) in a way that we can use a similar idea of PageRank, which is referred to as *VisualRank*. However, implementing *VisualRank* faces four challenges:

(1) The edges in the visualization link graph and the edges in the Webgraph have different meanings. In the Webgraph, an edge from a page X to a page Y if there exists a hyperlink on page X referring to page Y. In the visualization link graph, the edges have diversified semantics (or facets).
(2) The search behaviors between a normal search engine and a visualization search engine are quite different.
(3) We also want to use the graph in faceted navigation to reduce the number of interactions with the user. It is challenging to design multi-goal optimization algorithms.
(4) Our end goal is not to rank nodes in the visualization link graph, but the unlinked visualizations for an input dataset, for which we need to infer from the "importance values" of their matched good visualizations to rank.

Hence, a good research opportunity is how to design a *VisualRank* algorithm, which shares the basic intuition of PageRank, but serves the purpose of ranking visualizations.

3.2.4 Faceted Navigation. The recent proposal towards visualization recommendation systems [15] made an attempt to define the criteria of recommending good visualizations: relevance, surprise, non-obviousness, diversity and coverage. Another recent work proposes a general-purpose query language for visualization recommendation [18].

Opportunity. We can certainly use these criteria to create our facets. Unfortunately, these are just research hypothesis – whether they can lead to good visualizations is still not well justified. Besides, there is no well accepted implementation for each of them. Hence, it remains open that which facets should be considered and how to implement them.

3.3 Optimization

Response time is critical to real-time applications – response times under one second are usually considered to be good for search engines. Naturally, this requires us to carefully design DeepEye, from storage, indexes, to algorithms.

Opportunity. The first opportunity is about storage: What is the best physical representation of the data and the existing visualizations, assuming that they are stored in tables and JSON files? The work RodenStore [8] proposed an adaptive and declarative storage system providing a high-level interface for describing the physical representation of data. The similar idea can be adopted in the problem of storing datasets and visualizations. The second opportunity is how to store the visualization link graph – this will be tightly coupled with all (ranking) algorithms that use this graph. We can also group the visualizations effectively, *e.g.,* in a hierarchy or using partial orders, to help users to quickly find target visualizations. The third challenge is how to design effective index, especially for large datasets. We can borrow the idea from the search engine, *e.g.,* inverted index and forward index. However, it is rather challenging to design indexes for visualization search and ranking, because they involve more data features and more visualization operations.

4 RELATED WORK

Visualization Recommendation Systems. One important line of work is visualization recommendation systems [14–16]. Roughly speaking, given a dataset, they want to automatically recommend visualizations to the user under different criteria. DeepEye differs from them in the following two key aspects: (1) instead of guessing the user's intent, DeepEye accepts keyword search; and (2) DeepEye uses existing good visualizations to reason about the input dataset.

Interactive Data Visualization. There are some interactive data visualization systems, such as D3 [5], protovis [4], matplotlib [10], Tableau [17]. We do not plan to reinvent the wheel – DeepEye will export the user selected visualizations to the above toolkits for more complicated manipulations.

Visualization Languages. There are many visualization languages, *e.g.,* Vega [13], Vega-Lite [12], VizQL [9], Ermac [19], and DeVIL [20]. We use Vega-Lite as it is simple yet general enough.

5 CONCLUDING REMARKS

With increasing interest and importance of data visualization for data science applications, there is an emerging need for a tool to easily create good visualizations. We believe that DeepEye, the first visualization search engine, will lead to interesting and impactful problems for many communities, including: database, information retrieval, machine learning, and visualization.

Acknowledgement. This work was supported by 973 Program of China (2015CB358700), NSF of China (61632016, 61472198, 61521002, 61661166012), and TAL education.

REFERENCES

[1] I. Androutsopoulos, G. Ritchie, and P. Thanisch. Natural language interfaces to databases âĂŞ an introduction. *Natural Language Engineering*, 1(1):29âĂŞ81, 1995.
[2] Y. Bengio, A. C. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, 2013.
[3] C. Binnig, L. D. Stefani, T. Kraska, E. Upfal, E. Zgraggen, and Z. Zhao. Toward sustainable insights, or why polygamy is bad for you. In *CIDR*, 2017.
[4] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE Trans. Vis. Comput. Graph.*, 15(6):1121–1128, 2009.
[5] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE Trans. Vis. Comput. Graph.*, 17(12):2301–2309, 2011.
[6] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Seventh International World-Wide Web Conference (WWW 1998)*, 1998.
[7] C. J. C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. N. Hullender. Learning to rank using gradient descent. In *ICML*, pages 89–96, 2005.
[8] P. Cudré-Mauroux, E. Wu, and S. Madden. The case for rodentstore: An adaptive, declarative storage system. In *CIDR*, 2009.
[9] P. Hanrahan. Vizql: a language for query, analysis and visualization. In *SIGMOD*, page 721, 2006.
[10] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science and Engineering*, 9(3):90–95, 2007.
[11] F. Li and H. V. Jagadish. Understanding natural language queries over relational databases. *SIGMOD Record*, 45(1):6–13, 2016.
[12] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE Trans. Vis. Comput. Graph.*, 23(1):341–350, 2017.
[13] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE Trans. Vis. Comput. Graph.*, 22(1):659–668, 2016.
[14] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. G. Parameswaran. Effortless data exploration with zenvisage: An expressive and interactive visual analytics system. *PVLDB*, 10(4):457–468, 2016.
[15] M. Vartak, S. Huang, T. Siddiqui, S. Madden, and A. G. Parameswaran. Towards visualization recommendation systems. *SIGMOD Record*, 45(4):34–39, 2016.
[16] M. Vartak, S. Rahman, S. Madden, A. G. Parameswaran, and N. Polyzotis. SEEDB: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13):2182–2193, 2015.
[17] R. M. G. Wesley, M. Eldridge, and P. Terlecki. An analytic data engine for visualization in tableau. In *SIGMOD*, pages 1185–1194, 2011.
[18] K. Wongsuphasawat, D. Moritz, A. Anand, J. D. Mackinlay, B. Howe, and J. Heer. Towards a general-purpose query language for visualization recommendation. In *HILDA@SIGMOD*, page 4, 2016.
[19] E. Wu, L. Battle, and S. R. Madden. The case for data visualization management systems. *PVLDB*, 7(10):903–906, 2014.
[20] E. Wu, F. Psallidas, Z. Miao, H. Zhang, and L. Rettig. Combining design and performance in a data visualization management system. In *CIDR*, 2017.